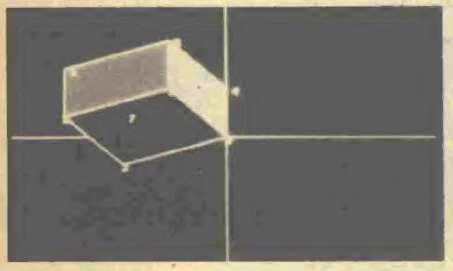
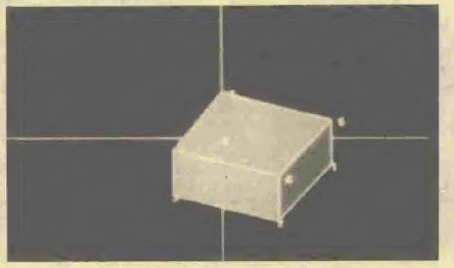
หลักการของ
กราฟิก 3 มิติ

ดร.ไพศาล สงวนหมู่

คอมพิวเตอร์กราฟิกเป็นเรื่องซึ่งกำลังได้รับการกล่าวถึงอย่างมากในขณะนี้ เนื่องจากว่าคอมพิวเตอร์ซึ่งสามารถทำกราฟิกได้อย่างละเอียดนั้นราคาถูกลง จนสามารถซื้อหามาใช้ในสำนักงานเล็ก ๆ เมื่อก่อนนี้กราฟิกที่มีความละเอียดสูง สามารถทำได้บนเครื่องที่มีความสามารถสูงเท่านั้น แต่เดี๋ยวนี้วิวัฒนาการทางด้านไมโครโปรเซสเซอร์ ทำให้เครื่องไมโครคอมพิวเตอร์มีขีดความสามารถใกล้เคียงกับมินิคอมพิวเตอร์

ทำไมการสร้างภาพกราฟิกจึงต้องใช้เครื่องที่มีความสามารถสูงท่านอาจจะถาม ก่อนอื่นขอบอกกล่าวเสียก่อนว่า การทำรูปกราฟิกนั้นอาจจะทำได้สองทางคือ ลักษณะที่เรียกว่าบิตแม็พ (Bit Map) หนึ่ง และวิธีที่เรียกว่าเวกเตอร์ (Vector) อีกหนึ่ง วิธีการของบิตแม็พนั้นเพียงแต่นำเอาอิมเมจของภาพนำมาเสนอออกที่หน้าจอ แต่ละจุดบนจอมาจากข้อมูลของภาพ 1 ตำแหน่งอาจจะเป็น 1 บิต หรือมากกว่า 1 บิตก็ได้ แต่แทนจุดการแสดงบนจอ 1 จุดหรือพิกเซล ลักษณะการแสดงกราฟิกแบบนี้ เราไม่สามารถที่จะเคลื่อนไหวรูปกราฟิกโดยมุมมองที่แตกต่างกันได้ จะพูดไปก็คือการนำเอาภาพนิ่งมาให้คอมพิวเตอร์แสดงเท่านั้นเอง ใส่ชีวิตชีวาให้โลดเต้นไม่ได้ ยกตัวอย่างเช่น โปรแกรมพวก PC PAINT, STORY BOARD เป็นต้น สำหรับวิธีที่สองนั้น การเก็บข้อมูลของภาพเป็นการเก็บแบบที่เรียกว่าข้อมูลของวัตถุนั้นจริง ๆ โดยจะมองวัตถุเหมือนกับว่าประกอบด้วยการลากเส้นจากจุดหนึ่งไปอีกจุดหนึ่ง จุดที่เส้นตรงลากมาพบกัน (อาจจะมากกว่า 2 เส้น) เรียกว่า เวอร์เท็กซ์ (Vertex) กำหนดรูปกราฟิกแบบนี้ ทำให้คอมพิวเตอร์ต้องใช้ความสามารถในการคำนวณสูงเพื่อที่จะนำจุดเวอร์เท็กซ์ต่างๆ มากำหนดลงในระนาบของการมองเห็นเพื่อสร้างเป็นภาพกราฟิกขึ้นมา จะเห็นว่าถ้าเราเปลี่ยนมุมมองใหม่ คอมพิวเตอร์ก็จะสามารถคำนวณตำแหน่งของจุดเวอร์เท็กซ์บนระนาบของการมองเห็นใหม่ได้ ดังนั้นกราฟิกชนิดนี้จึงมีชีวิตชีวาหมุนดูมุมมองต่าง ๆ ได้ ตัวอย่างเช่น โปรแกรมพวก AutoCAD, Energraphic เป็นต้น คราวนี้คงจะพอเข้าใจแล้วว่าทำไมคอมพิวเตอร์ที่มีความสามารถในการคำนวณสูงจึงมีความสามารถในการทำกราฟิกได้สูง ไม่เห็นตรงกับคนเลยนะครับ คนที่มีความสามารถในการคำนวณสูงส่วนมากจะขาดความสามารถในทางด้านศิลปะ

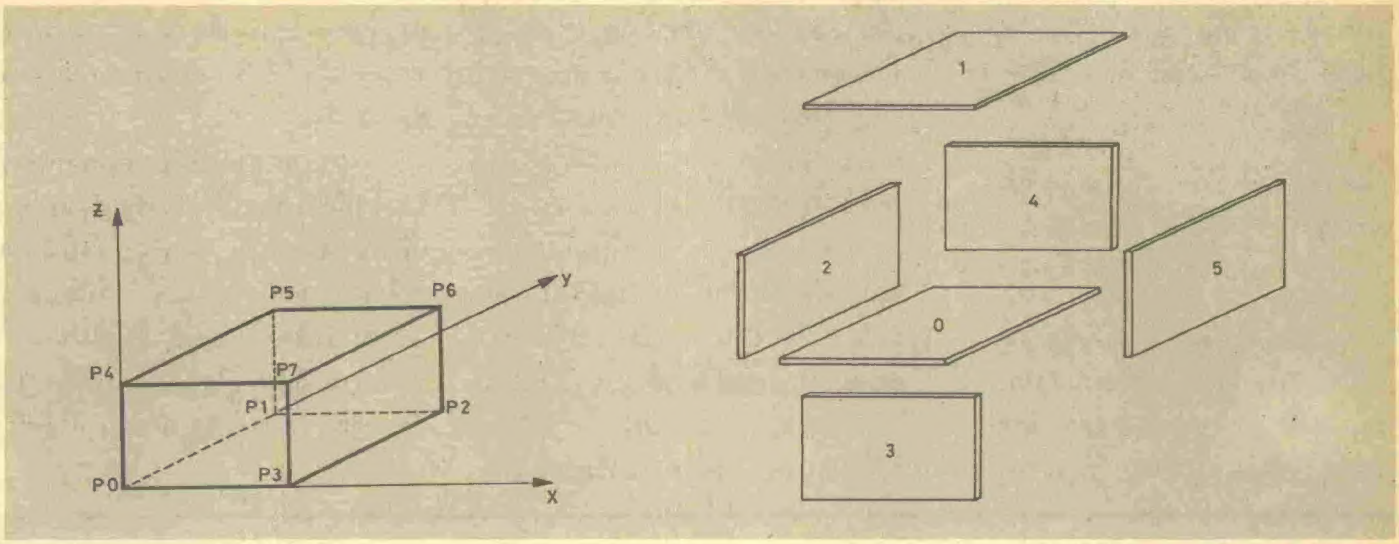
เรื่องที่ผมจะนำมากล่าวไว้ในที่นี้ก็คือเรื่องของหลักการทำภาพสามมิติเอากันแบบง่าย ๆ ไม่ใช้ศัพท์แสงทางคณิตศาสตร์มากนัก พร้อมกับมีตัวอย่างการเขียนโปรแกรมไว้ทดสอบหลักการที่ว่าไว้ด้วย โดยจะกล่าวถึงทฤษฎีของการมองเห็น และหลักการของสร้างภาพกราฟิกว่าส่วนไหนของวัตถุที่สามารถมองเห็นและมองไม่เห็น เมื่อเราหมุนมุมมองวัตถุไป

โพลีกอน (รูปหลายเหลี่ยม)

ถ้าท่านเคยพบสเปคของเครื่องคอมพิวเตอร์กราฟิก ท่านคงจะเคยเห็นสรรพคุณที่เขาบอกไว้ว่า มีความสามารถในการเขียนภาพ เท่านั้นเท่านี้ โพลีกอนใน 1 วินาที เรามาทำความเข้าใจกับคำว่า โพลีกอน (Polygon) กันก่อนดีไหม?

โพลีกอน (Polygon) ก็คือรูปหลายเหลี่ยมตั้งแต่สามเหลี่ยมขึ้นไป (สองเหลี่ยมคงจะเป็นรูปร่างไม่ได้) จะเป็นกี่เหลี่ยมก็แล้วแต่ไม่จำกัด ผิวของวัตถุประกอบด้วยรูปหลายเหลี่ยม หลายรูปประกอบกัน การสร้างรูปกราฟิกของวัตถุก็เช่นกัน ประกอบด้วยรูปหลายเหลี่ยมหลายรูป ดังนั้นรูปหลายเหลี่ยมหรือโพลีกอนที่กล่าวถึงก็คือ ส่วนที่เล็กที่สุดของพื้นผิวการมองเห็น ถ้าท่านสังเกตภาพกราฟิกที่สร้างมาจากคอมพิวเตอร์จะเห็นว่าประกอบด้วยโพลีกอนมากมาย ยิ่งจำนวนโพลีกอนยิ่งมาก ยิ่งทำให้กราฟิกนั้นใกล้เคียงภาพจริงมากขึ้นเท่านั้น

โพลีกอน ประกอบด้วยอะไรบ้าง โพลีกอนประกอบขึ้นจากการลากเส้นตรงจากจุดเริ่มต้นผ่านจุดต่าง ๆ บนระนาบเดียวกันจนกลับมายังจุดเริ่มต้นอีกครั้งหนึ่ง ดังนั้น โพลีกอนสามเหลี่ยม



รูปที่ 1 กำหนดจุดต่าง ๆ ในรูปร่างของวัตถุและโพลีกอนที่ประกอบกันเป็นรูปร่าง

ประกอบไปด้วยจุด 4 จุด ที่เส้นตรงจะลากผ่านไปโดยจุดเริ่มต้นและจุดสุดท้ายเป็นจุดเดียวกัน

การกำหนดลักษณะของวัตถุ

การกำหนดรูปร่างของวัตถุ เราอาจจะบอกโดยจุดโคออร์ดิเนตของจุดในแกน xyz ในตัวอย่างที่จะกล่าวถึงต่อไปนี้ สมมติว่าเรามีก้อน 4 เหลี่ยมก้อนหนึ่งดังรูปที่ 1 ในก้อน 4 เหลี่ยมก้อนนี้ประกอบด้วยจุดตัดที่เป็นเวอร์เท็กซ์ทั้งหมด 8 จุด คือ P_0, P_1, P_2, P_3, P_4, P_5, P_6 และ P_7 สมมติว่า เรากำหนดจุดต่าง ๆ ในแกน xyz เป็นอัตราส่วนกันดังนี้

P0 = (0,0,0,)
P1 = (0,2,0)
P2 = (2,2,0)
P3 = (2,0,0)
P4 = (0,0,1)
P5 = (0,2,0)
P6 = (2,2,1)
P7 = (2,0,1)

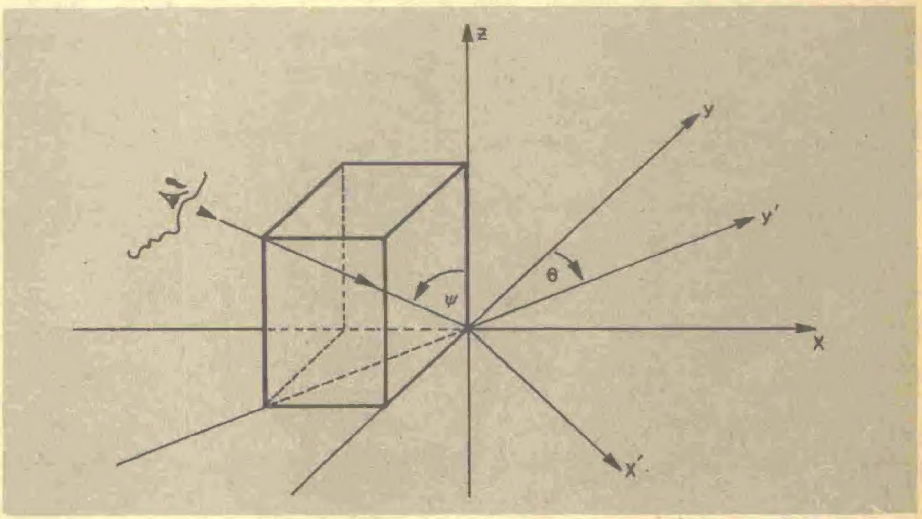
นอกจากกำหนดจุดต่าง ๆ ในรูปร่างของวัตถุแล้ว เราจะต้องกำหนดพื้นผิวของวัตถุในลักษณะของโพลีกอนอีกด้วย สมมติก้อนสี่เหลี่ยมนี้เป็นผิวตันหมดไม่ใช่กล่องที่เปิดฝาด้านใดด้านหนึ่ง ดังนั้นเราจะมีโพลีกอนทั้งหมด 6 ด้าน คือ Poly 0 ถึง Poly 5 โดยกำหนดจุดในโพลีกอนดังนี้

Poly [0] = (P_0, P_1, P_2, P_3, P_0)
Poly [1] = (P_4, P_5, P_6, P_7, P_4)
Poly [2] = (P_0, P_1, P_5, P_4, P_0)
Poly [3] = (P_0, P_4, P_7, P_3, P_0)
Poly [4] = (P_1, P_5, P_6, P_2, P_1)
Poly [5] = (P_3, P_7, P_6, P_2, P_3)

แน่นอนการกำหนดรูปร่างของวัตถุนี้อาจจะนำเข้าสู่คอมพิวเตอร์ได้หลายทาง เป็นต้นว่าการป้อนเข้าไปทางคีย์บอร์ด การอ่านมาจากแฟ้มหรือว่าการอ่านมาจากดิจิไตเซอร์

มุมของการมองและระนาบของการมองเห็น

หลักการง่าย ๆ ของการมองวัตถุก็คือว่า เราจะมองวัตถุในมุมมองไหน สมมติว่าเรากำหนดมุมมองออกเป็น 2 ส่วนคือ ψ มุมที่เอียงมาจากแกน z และ θ มุมที่เอียงมาจากแกน y ดังในรูปที่ 2



รูปที่ 2 มุมของการมอง

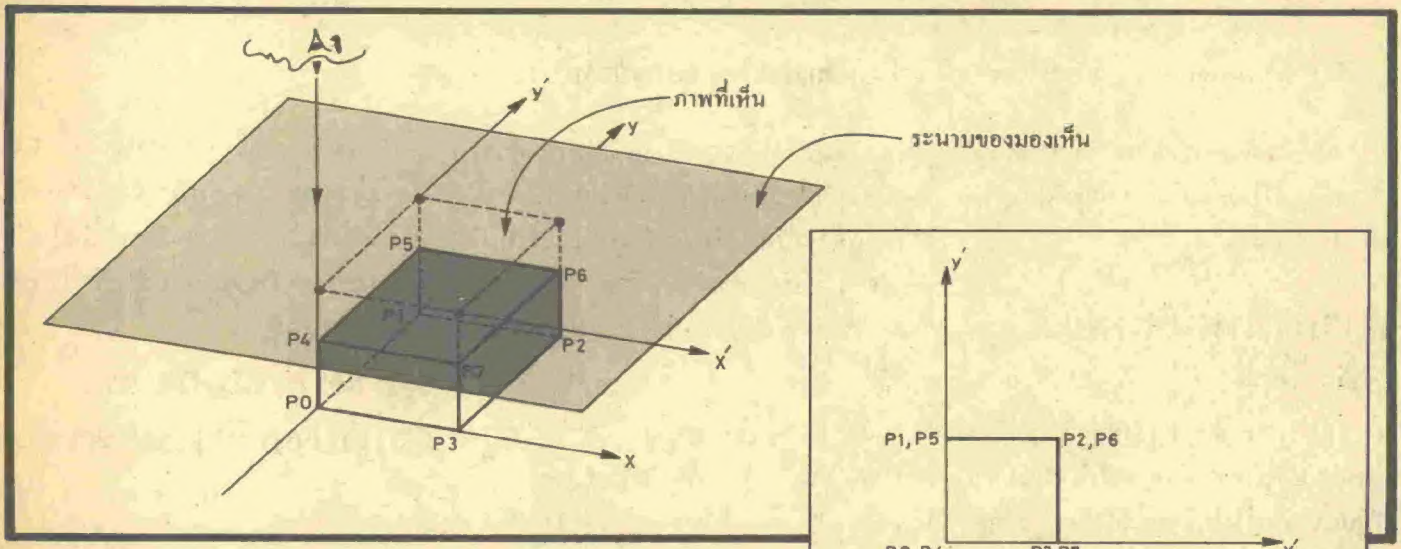
ถ้าเรามองโดยมุม $\psi = 0$ นั่นก็คือ เรามองที่ Top View ของวัตถุ

ระนาบของการมองเห็นก็คือ ระนาบที่ตั้งฉากกับทิศทางของการมองจุด Vertex ต่าง ๆ ของวัตถุ ที่เราจะมองเห็นก็คือจุดที่ปรากฏบนระนาบของการมองเห็น

เรามาลองดูง่าย ๆ ก่อน โดยเรามองที่ Top View หรือมุมมองในแกน z เป็น 0 ระนาบของการมองเห็นก็คือระนาบที่ขนานกับระนาบ xy เราจะมองเห็น Top View วัตถุดังในรูปที่ 3 จุดทุกจุดจะยังคง Project ลงบนระนาบของการมองเห็นในตำแหน่งเดียวกับระนาบ xy

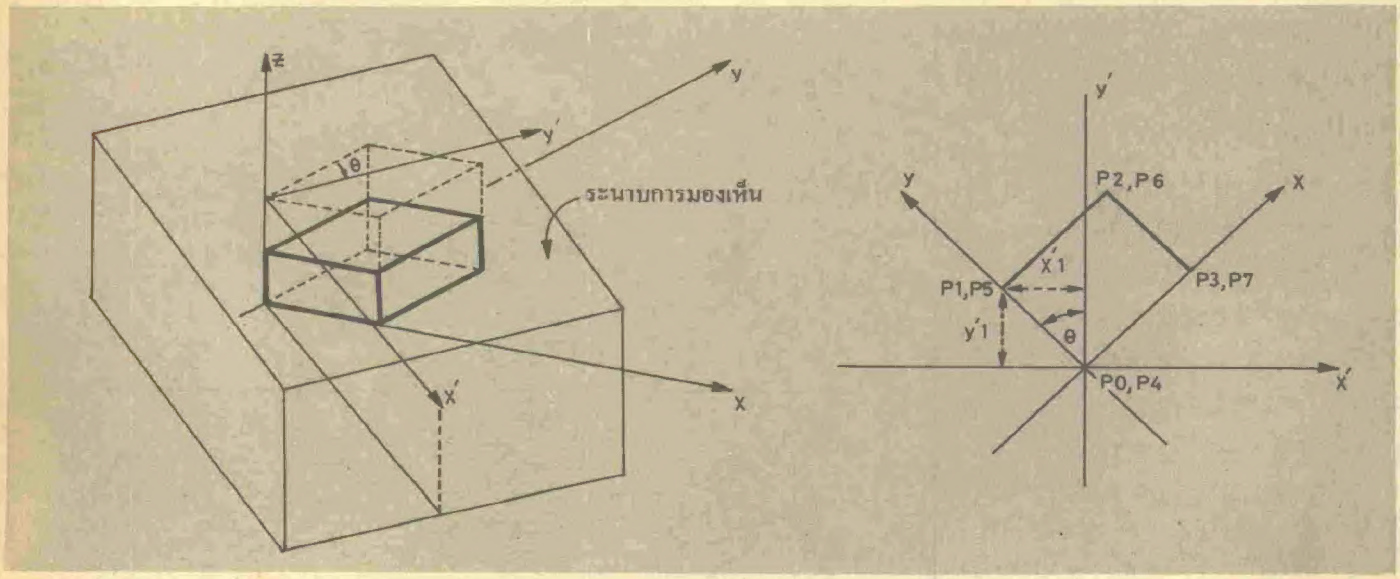
คราวนี้มาดูว่า ถ้าเราหมุนมุมมองในแกน y ออกไป โดยยังมองใน Tap View อยู่ สิ่งที่ปรากฏออกมาก็คือ การหมุนตำแหน่งของวัตถุนั้นเอง โดยจุดโคออร์ดิเนตของจุดเวอร์เท็กซ์ต่าง ๆ ย้ายไปยังแกนใหม่ โดยหลักเกณฑ์ของการย้ายแกนทางเรขาคณิต ทำให้เราสามารถหาจุดต่าง ๆ ที่ปรากฏบนระนาบของการมองเห็นได้

การย้ายแกนทำได้ง่ายในทางโพลาร์โคออร์ดิเนต โดยมีรัศมีคงที่ เพียงแต่มุมเท่านั้นที่เปลี่ยนไป ดังนั้น ถ้าหากเราเปลี่ยนมุมมองไป แกน y ไปเป็นมุม 0 ตามเข็มนาฬิกา วัตถุที่ปรากฏบนระนาบของการมองเห็นก็คือ การหมุนวัตถุไปทิศทางทวนเข็มนาฬิกาเป็นมุม 0 ดังในรูปที่ 4



รูปที่ 3ก. ระนาบของการมองเห็น $\psi = 0$ และ $\theta = 0$

รูปที่ 3ข. ภาพที่มองเห็น บนระนาบของการมองเห็น



รูปที่ 4 ระนาบการมองเห็นและภาพที่มองเห็นบนระนาบของการมองเห็น

โคออร์ดิเนตของจุดต่าง ๆ จะปรากฏบนระนาบการมองเห็น xy′ดังนี้

$$X' = \sqrt{x^2+y^2} * \sin(\tan^{-1}\frac{x}{y} - \theta)$$
$$Y' = \sqrt{x^2+y^2} * \cos(\tan^{-1}\frac{x}{y} - \theta)$$
$$Z' = Z$$

เทอม $\sqrt{x^2+y^2}$ ก็คือรัศมีในโพลาร์โคออร์ดิเนต

เทอม $\tan^{-1}\frac{x}{y}$ ก็คือมุมในโพลาร์โคออร์ดิเนต

ในการหมุนโพลาร์โคออร์ดิเนตไปเป็นมุม θ เราจะได้ x, y ใหม่ จากการเปลี่ยนจากโพลาร์โคออร์ดิเนตโดยที่รัศมีเท่าเดิมแต่มุมเปลี่ยนไปด้วยค่า θ

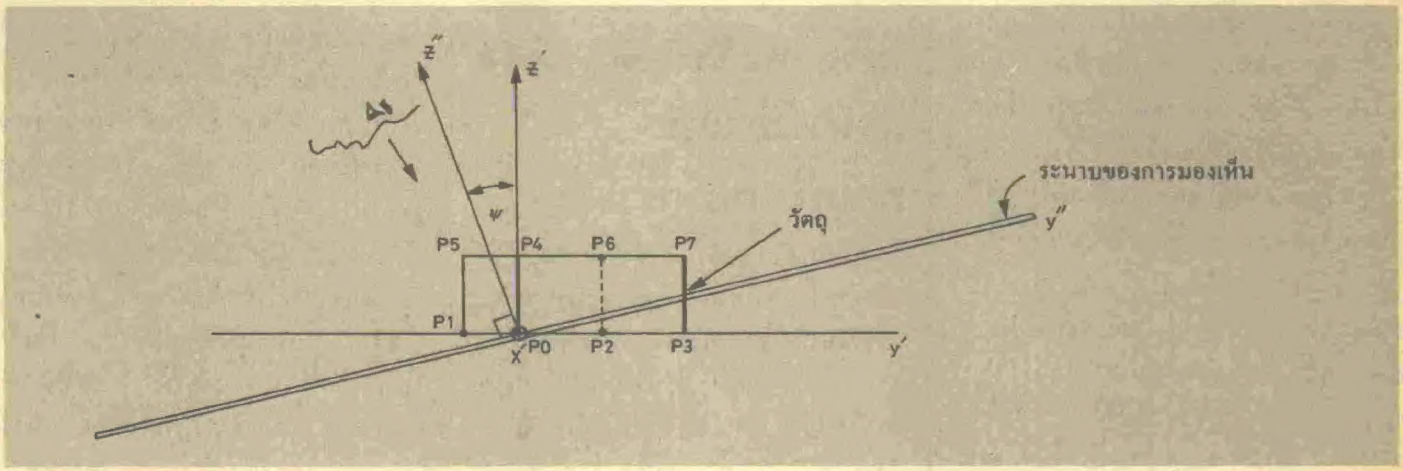
คราวนี้มาลองเปลี่ยนมุมมองในแนวแกน z ดูซิว่าอะไรจะเกิดขึ้น สมมติว่าเราเอียงระนาบของการมองในแนวที่ขนานกับแกน x′โดยมุมมองของ z เปลี่ยนไป ก่อนอื่นขออนุญาตย้ายระนาบของการมองลงไปยังจุด origin เดิมเสียก่อน เพื่อความเข้าใจได้ง่ายเข้า ทุกอย่างยังถูกต้องเหมือนเดิมใช่ไหมครับ เอาละคราวนี้เริ่มเอียงระนาบของการมองเห็นไปเป็นมุม ψ เมื่อเทียบกับแกน z′เพื่อความเข้าใจง่าย และง่ายต่อการเขียนภาพ ถ้าเราเขียนภาพตั้งฉากกับแกน x′จะเห็นภาพของการมองดังในรูปที่ 5

จากนี้เราจะพบว่าเงาหรือโปรเจคชันของจุดต่าง ๆ ที่ปรากฏบนระนาบของการมองเห็นที่เคยทับกันอยู่จะเริ่มแยกออกจากกันให้เห็นชัดเจนโคออร์ดิเนตของเวอร์เท็กซ์ที่ปรากฏบนระนาบของการมองเห็นจะเปลี่ยนไป จาก x′, y′ไปเป็น x″, y″ ความสูงของวัตถุหรือโค–ออร์ดิเนตในแกน z เริ่มแสดงผลต่อการมองเห็น มาดูรายละเอียดในรูปที่ 6

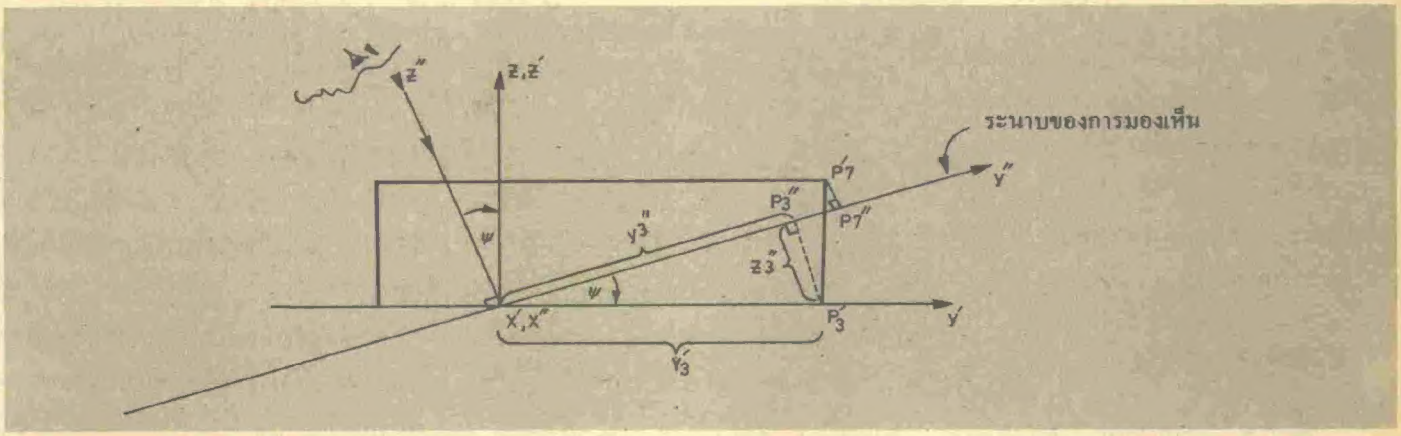
มาลองพิจารณาดู 2 จุด P3′และ P7′จะพบว่าเงาของ P3′และ P7′ปรากฏบนระนาบของการมองเห็นเป็น P3″ และ P7″ ดังนั้นโคออร์ดิเนตของจุดที่ปรากฏบนระนาบของการมองเห็นคือ

$$x'' = x'$$
$$y'' = y'*\cos\psi + z'*\sin\psi$$

จุด P3′และ P7′มีโคออร์ดิเนต y′ที่เหมือนกันแต่ z′ไม่เท่ากัน P7 มี z′มาก



รูปที่ 5 ภาพตัดตั้งฉากกับแกน x′แสดงมุมมองในแกน z ที่เปลี่ยนไป



รูปที่ 6 ภาพตัดในระนาบ y″.z″

กว่า ดังนั้นจะปรากฏบนระนาบของการมองเห็น ในแกน y″ ที่สูงกว่า P3

รวมผลของการย้ายมุมมอง

ถ้ารวมผลการหมุนมุมมองทั้งในแนวแกน y และในแนวแกน z เข้าด้วยกัน จะพบว่าการย้ายเงาของแต่ละจุดไปยังระนาบของการมองเห็นเป็นดังนี้

$$
\begin{aligned}
x'' &= \sqrt{x^2 + y^2} * \sin(\tan^{-1}x/y - \theta) \\
y'' &= \sqrt{x^2 + y^2} * \cos(\tan^{-1}x/y - \theta) * \cos\psi + z * \sin\psi \\
z'' &= z\cos\psi - \sqrt{x^2y^2} * \cos(\tan^{-1}x/y - \theta) * \sin\psi
\end{aligned}
$$

เราก็จะได้จุดทุกจุดบนระนาบของการมองเห็นแล้ว คราวนี้ก็มาถึงปัญหาที่สำคัญละครับว่า โพลีกอนหรือพื้นผิวไหนบังพื้นผิวไหนบ้าง ผิวไหนที่มองเห็นหรือผิวไหนที่มองไม่เห็นบ้าง มีผลงานการวิจัยหลายชิ้นบอกถึงอัลกอริทึมที่บ่งบอกถึงหนทางการตัดสินใจว่า พื้นผิวไหนบังผิวไหนโดยใช้เวลาในการตัดสินใจน้อยที่สุด ในที่นี้จะกล่าวถึงวิธีการง่าย ๆ ตามหลักการของการมองเห็น

หลักการง่าย ๆ ก็คือว่า พื้นผิวที่อยู่ใกล้สายตากว่า ย่อมจะบังพื้นผิวที่อยู่ไกลกว่า ถ้าหากพื้นผิวทั้งสองซ้อนกันอยู่ อ๊ะพูดง่ายดีนี่ ครับง่ายดี ก็ให้น้ำหนักของระยะทางในแกน z″ ของโพลีกอนซะ โดยการรวมระยะทางในแกน z″ ทั้งหมดของโพลีกอนแล้วหารด้วยจำนวนจุดที่ประกอบเป็นโพลีกอน ก็จะเป็นระยะทางเฉลี่ยในแนวแกน z″ ที่มีค่าสูงสุดก็คือพื้นผิวที่อยู่ใกล้สายตามากที่สุด (เพราะเรามองในแนวแกน z″)

การแสดงภาพ

เมื่อเราได้จุดทุกจุดบนระนาบของการมองเห็นรวมทั้งระดับความสำคัญของพื้นผิวแล้ว ก็สามารถนำมาแสดงบนจอภาพได้อย่างถูกต้อง โดยการเริ่มเขียนพื้นผิวที่อยู่ไกลสุดก่อน ก็คงจะต้องมีการเรียงลำดับความสำคัญของพื้นผิวเสียก่อนโดยการนำเอาระยะทางเฉลี่ยของแกน z″ ของแต่ละพื้นผิวมาเรียงลำดับตั้งแต่น้อยไปหามาก การแสดงภาพโพลีกอนก็ไล่ตามลำดับที่เรียงเอาไว้ เพราะฉะนั้นพื้นผิวที่อยู่ไกลสุดจะถูกเขียนก่อน ถ้าหากมีพื้นผิวไหนอยู่ใกล้กว่าก็จะถูกเขียนทับรูปเดิมเป็นการแก้ปัญหาเรื่องพื้นผิวที่มองเห็นและมองไม่เห็นไปในตัว

มีเรื่องอะไรอีกที่จะต้องศึกษาเกี่ยวกับคอมพิวเตอร์กราฟิก

ที่กล่าวมาทั้งหมดนี้เป็นเพียงส่วนเริ่มต้นที่เล็กน้อยเกี่ยวกับคอมพิวเตอร์กราฟิกเท่านั้น ยังมีเรื่องที่จะศึกษากันอีกมากสำหรับผู้ต้องการเขียนโปรแกรมทำกราฟิก และก็ยังมีเรื่องที่จะต้องทำการศึกษาวิจัยในระดับวิทยานิพนธ์ของปริญญาเอกอีกมากมาย ยกตัวอย่างเช่น การให้สีกับวัตถุที่มีเงาของแสงมาเกี่ยวข้อง เรื่องของการสร้างเงาบนผิว เมื่อมีแหล่งกำเนิดของแสงมากกว่า 1 แหล่งอยู่ในตำแหน่งที่ต่างกัน เรื่องของเปอร์สเปคตีฟ เรื่องของการนำวัตถุเคลื่อนที่ในลักษณะที่เรียกว่า Animation เรื่องของการเก็บข้อมูลของวัตถุในลักษณะที่สิ้นเปลืองเนื้อที่น้อยที่สุด เรื่องของลักษณะพื้นผิวที่สามารถแทนได้ด้วยสมการทางคณิตศาสตร์ เพื่อการประหยัดจำนวนโพลีกอน โฮ้ย...อีกมากมาย รับรองได้ว่ามีเรื่องให้ทำวิทยานิพนธ์ตั้งแต่ปริญญาโท ถึงปริญญาเอกสบายสบาย แค่ศึกษาผลงานที่เขาศึกษากันมาสิบกว่าปีมานี้ก็แย่แล้ว ท่านที่สนใจจริง ๆ ลองหาหนังสือที่ ดร.กรรชิต มาลัยวงศ์ แนะนำเอาไว้ในวารสารไมโครฉบับที่ 39 มาอ่านดูซิครับ หรือว่าจะลองหาวารสารของ ACM ในเรื่องของ Computer Graphic โดยเฉพาะซึ่งออกทุก ๆ เดือน อีกทั้งมี Computer Graphics Proceeding รวมเอาผลงานการวิจัยทางด้านนี้ออกมาทุก ๆ ปี ปีละ 4 เล่ม

ตัวอย่างการเขียนโปรแกรม

จากความรู้เบื้องต้นเกี่ยวกับภาพสามมิติที่เราได้มา ลองมาเขียนโปรแกรมทดสอบดูซิว่าเป็นดังว่าจริงฤๅ ภาษาที่ใช้เป็นเครื่องมือทดสอบคือ เทอร์โบปาสคาล เวอร์ชัน 4.0 แม้ว่าผมจะชอบภาษาซีมากกว่าปาสคาลก็ตาม คราวนี้จำยอมใช้เทอร์โบปาสคาล เวอร์ชัน 4.0 เนื่องจากว่าทดสอบได้รวดเร็ว คือเขียนแล้วดูได้ทันทีว่าผิดตรงไหน และมีคำสั่งเกี่ยวกับกราฟิกให้ใช้เยอะมาก แม้ว่าไมโครซอฟต์ซี เวอร์ชัน 5.0 จะมีคำสั่งมากพอ ๆ กัน แต่การเริ่มต้นในการเขียนโปรแกรมค่อนข้างจะลำบาก ไม่เหมือนกับเทอร์โบปาสคาล เวอร์ชัน 4.0 จับแผ่นใส่ช่อง A ก็สามารถทำงานได้ทันที อ้าว...กลายเป็นเรื่องการวิจารณ์ภาษาไปโน่น กลับมาเรื่องตัวอย่างโปรแกรมอีกหน อย่างที่ผมบอกเอาไว้ข้างบนว่าผมไม่ค่อยชอบปาสคาลเท่าไหร่ ดังนั้นถ้าการใช้คำสั่งยังไม่ดีพอก็ขออภัยไว้ด้วย แต่ขอให้ศึกษาอัลกอลิทึมก็แล้วกัน

ตัวโปรแกรมไม่ทำอะไรมาก นอกจากวาดกราฟสามมิติของแท่งสี่เหลี่ยมอันหนึ่งที่มุมมองต่าง ๆ กัน การเปลี่ยนมุมมองในแนวแกน y ทำได้โดยการ กด

แป้นควบคุมเคอร์เซอร์ UP เป็นการเพิ่มมุมมองในแกน y 5 องศา, Down เป็นการลดมุมมองในแกน y 5 องศา ในทำนองเดียวกัน RIGHT และ LEFT สำหรับเพิ่มและลดมุมมองในแนวแกน z ทีละ 5 องศา เมื่อลดหรือเพิ่มมุมมองแล้ว โปรแกรมจะทำการคำนวณจุดที่ตกไปยังระนาบของการมองเห็นใหม่ เสร็จแล้วคำนวณหาระยะทางในแนวแกน z″ เฉลี่ยของแต่ละโพลีกอน เพื่อนำมาใช้ในการกำหนดว่าโพลีกอนอันไหนควรจะเขียนก่อน โดยระยะเฉลี่ยของแต่ละโพลีกอนจะนำมาเรียงลำดับจากน้อยไปหามาก เมื่อได้ลำดับแล้วก็นำโพลีกอนมาเขียนบนจอภาพ

ในส่วนหัวของโปรแกรมต้องไม่ลืม USES Statement ด้วย เป็นการบอกเทอร์โบปาสคาลว่า ให้รวมเอาส่วนที่เป็นกราฟ (ซึ่งอยู่ในแฟ้ม Graph.TPU) มาใช้ในการคอมพายล์ด้วย อันนี้สำคัญเพราะถ้าไม่มี (คนขับกราฟ?) Graph Driver การคอมพายล์จะไม่ผ่าน

โปรแกรมเริ่มต้น โดยการกำหนดวัตถุโดยการใช้ constant กำหนดจุด x,y,z ของวัตถุ โดยใช้ชื่อ constant ที่เป็นอะเรย์ว่า xpoint, ypoint และ zpoint นอกจากกำหนดจุดต่าง ๆ ของวัตถุแล้ว เราจะต้องบอกด้วยว่าวัตถุนี้มีพื้นผิวโพลีกอนอะไรบ้าง และแต่ละโพลีกอนประกอบด้วยจุดไหนบ้าง ในที่นี้เราจะมีโพลีกอนอยู่ 6 ด้าน โปรแกรมในบรรทัดที่ 17 จะบอกว่ามีโพลีกอน 6 ตัว แต่ละตัวประกอบด้วยจุด 5 จุด ส่วนบรรทัดที่ 20 เป็นการกำหนดวิธีการระบายสีเอาไว้ใช้(ในที่นี้ใช้ระบายแบบตาหมากรุก)

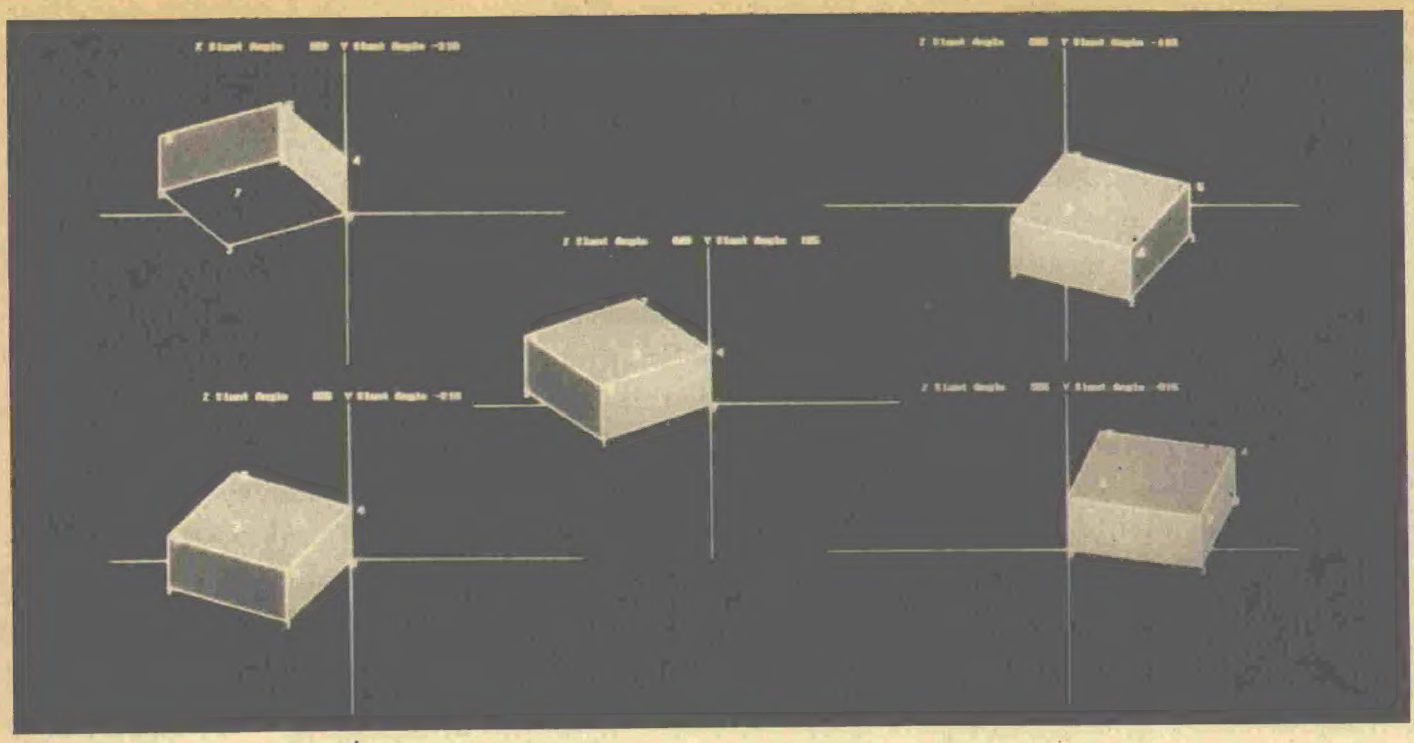
เรากระโดดข้ามโพรซีเดียว Draw และ SORT ไปก่อน ข้ามไปยัง Main Program ในบรรทัดที่ 136 เริ่มต้นด้วยการกำหนดเท็กซ์ที่จะพิมพ์บนจอกราฟิก แล้วการตรวจสอบกราฟิกการ์ดที่ต่อไว้กับระบบในบรรทัดที่ 138 ซึ่งจะให้ผลออกมาเป็นตัวแปร 2 ตัว คือ grdriver และ grmode ตัวแปรทั้ง 2 ตัวนี้ถูก declare ไว้แล้วใน Graph.TPU ค่าที่ได้ออกมาจะบอกว่าการ์ดที่ใช้อยู่เป็นการ์ดชนิดไหนและกำลังใช้โหมดไหนอยู่(ดูหนังสือคู่มือเทอร์โบปาสคาลเวอร์ชัน 4.0 ประกอบ) เนื่องจากการแสดงกราฟของโปรแกรมนี้ ใช้ความละเอียดสูงและต้องการแสดงสีด้วยถึง 6 สีอย่างน้อย จึงกำหนดไว้ว่าการ์ด EGA หรือ VGA เท่านั้นจึงจะให้ทำงานต่อ ดังนั้นบรรทัดที่ 140 ถึง 144 จึงนำมาใส่ไว้เพื่อไม่ให้เครื่องทำงานต่อ ถ้าไม่ใช้ EGA หรือ VGA ท่านผู้อ่านอาจจะเปลี่ยนแปลงเอาเอง แต่ว่าจะต้องเปลี่ยนแปลงค่า จุด x,y ที่ปรากฏบนจอเสียใหม่ให้สอดคล้องกับความสามารถในการแสดงกราฟิกของการ์ดนั้น ๆ เป็นต้นว่า CGA แสดงภาพได้ด้วยความละเอียดสูง 600 × 200 จุด ในลักษณะขาวดำ ในที่นี้เป็น EGA ความละเอียดของภาพใน grmode = 1 สำหรับการ์ดที่มีหน่วยความจำมากกว่า 64 กิโลไบต์ จะแสดงภาพได้ละเอียด 640 × 350 จุด แต่ละจุดแสดงภาพได้ 16 สี

โปรแกรมเริ่มต้นด้วยการเปลี่ยนสกรีนและตั้งค่าหน้าต่างของกราฟิกให้กำหนดอยู่ภายในกรอบอันหนึ่งในบรรทัดที่ 146 และ 147 กำหนดมุมมองเริ่มต้น Z angle และ Y angle เป็น 0

ลูปหลักของการทำงานอยู่ระหว่างบรรทัดที่ 150 ถึง 179 โดยเริ่มต้นจากการคำนวณจุดต่าง ๆ บนระนาบของการมองเห็นโดยใช้สูตรที่เราพูดถึงในตอนต้นของเรื่องนี้ บรรทัดที่ 153 และ 154 เป็นการหาค่า cos และ sin ของมุม y angle เสียก่อน เนื่องจากไม่ต้องการให้ไปคำนวณซ้ำแล้วซ้ำเล่า ในลูปบรรทัดที่ 156 ถึง 167 เนื่องจากค่าดังกล่าวเป็นค่าคงที่ภายในลูปดังกล่าวและมีการใช้ในบรรทัดที่ 165 และ 166 ถึงสองครั้ง โปรแกรมเมอร์ที่ดีควรจะคำนึงถึงเทคนิคนี้เอาไว้ด้วยเพื่อประหยัดเวลาในการคำนวณของเครื่อง เนื่องจากคำนวณ sin หรือ cos จะใช้เวลามาก ถ้ามีทางที่ลดการคำนวณค่าเก่า ซ้ำซากควรจะนำออกมาไว้นอกลูป

เมื่อคำนวณจุดต่าง ๆ ที่ฉายลงบนระนาบของการมองเห็นแล้วคราวนี้ก็มาคำนวณหาค่าเฉลี่ยของระยะทางในแกน z″ และนำมาเรียงลำดับกันละโพรซีเดียว Sort ทำหน้าที่ดังกล่าว การให้น้ำหนักในโปรแกรมไม่ได้เฉลี่ยเอาไว้เพราะทุกโพลีกอนมีจำนวนจุดเท่ากัน ถ้าหากจะนำไปใช้กับโพลีกอนหลายชนิดปนกันอยู่ อย่าลืมเอาจำนวนจุดไปหารด้วย เพื่อจะได้นำมาเปรียบเทียบกันได้อย่างถูกต้อง การเรียงตามลำดับใช้วิธีการไล่เปรียบเทียบธรรมดาเพื่อความเข้าใจง่าย ถ้าจะให้ดีก็นำการเปรียบเทียบแบบไบนารี่หรือวิธีการอื่น ๆ เพื่อที่จะทำให้การเรียงลำดับทำได้เร็วขึ้น เมื่อเรียงลำดับเสร็จแล้วเราจะได้ลำดับของการวาดโพลีกอนอยู่ในตัวแปรที่เป็นอะเรย์ชื่อว่า Index, Index[0] ก็จะเป็นหมายเลขโพลีกอนที่มีค่าเฉลี่ยของระยะทางแกน z″ น้อยที่สุด จะต้องนำมาเขียนก่อน

โพรซีเดียว Draw จะทำหน้าที่เขียนโพลีกอนและระบายสีโพลีกอนตามลำดับก่อนหน้าหลังตามที่เรียงเอาไว้ การเขียนและระบายโพลีกอนในเทอร์โบ-ปาสคาลเวอร์ชัน 4.0 ใช้คำสั่งเดียวคือ Fill Poly แต่จะต้องมีตัวแปรที่เป็นอะเรย์ของจุด x,y ให้กับคำสั่ง ในที่นี้ใช้ตัวแปรที่ชื่อ rectang ในส่วนเริ่มต้นของการวาด แต่ละโพลีกอนจึงจำเป็นต้องเอาค่าของแต่ละโพลีกอนกำหนดลงไปในตัวแปร rectang เสียก่อนในบรรทัดที่ 52 ถึง 58 บรรทัดที่ 56 และ 57 เป็นการกำหนดจุดโคออร์ดิเนตบนจอภาพด้วย เนื่องจากในจอภาพมีจุดโคออร์ดิเนตเริ่มจากมุมซ้ายบน ในที่นี้เราจะกำหนดให้



รูปที่ 7 แสดงการสร้างภาพที่มุมมองต่าง ๆ กัน

จุด origin อยู่ที่ตำแหน่ง X:250 และ Y:160 นอกจากนั้นยังกำหนดมาตราส่วนในแกน x และแกน y ให้เป็นสัดส่วนกันด้วย ในที่นี้กำหนดให้ 5 จุดในแกน y มีความยาวเท่ากับ 8 จุดในแกน x (ใกล้เคียงกับความจริง ถ้าจอภาพเป็น 4 เหลี่ยมจัตุรัส 640 จุดในแกน x จะให้ความยาวเท่ากับ 350 จุดในแกน y)

บรรทัดที่ 59 กำหนดรูปแบบของการระบายสีเป็นแบบตาหมากรุก

บรรทัดที่ 60 เขียนโพลีกอนและระบายสีด้วยสีหมายเลขตามลำดับของหมายเลขโพลีกอน

บทส่งท้าย

หวังว่า ผมคงจะมีโอกาสได้เปิดประตูให้ท่านได้ก้าวสู่การพัฒนา 3-D กราฟิกได้บ้าง แม้ว่าปัจจุบัน มีแพคเกจต่าง ๆ ซึ่งสามารถในการเขียนภาพ 3-D แล้วก็ตาม ถ้าเราไม่ลองเล่นดูจริง ๆ เราก็ไม่มีทางพัฒนาขึ้นมาเองได้ นอกจากใช้ของคนอื่นเขาพัฒนามาให้เรา (แอบ) ใช้เท่านั้น

ตัวอย่างโปรแกรมวาดรูปกราฟ 3 มิติของแท่งสี่เหลี่ยม

```
File name THREED.PAS   File Date 09-05-88  Print Date 10-10-88 Time 08:12

 1:  { ************************************************************
 2:    Three Dimension Graphic Demonstration program.
 3:
 4:    The solid rectangular object is defined, and then
 5:    a different of looking angle will give the different
 6:    3-D picture on the screen...
 7:    Dr.Paisan Sangaunmu  09/04/88
 8:    ************************************************************}
 9:  program graphic;
10:  USES
11:    graph,crt;
12:  const
13:  {*******define the rectangular solid object***}
14:  xpoint:array[0..7] of integer=(0,0,2,2,0,0,2,2);
15:  ypoint:array[0..7] of integer=(0,2,2,0,0,2,2,0);
16:  zpoint:array[0..7] of integer=(0,0,0,0,1,1,1,1);
17:  poly:array[0..5,0..4] of integer =((0,1,2,3,0),(4,5,6,7,4),
18:                                     (0,1,5,4,0),(0,4,7,3,0),
19:                                     (1,5,6,2,1),(3,7,6,2,3));
20:  gray1:fillpatterntype=($aa,$55,$aa,$55,$aa,$55,$aa,$55);
21:  Type
22:     Ptype  = array[0..1] of integer ;
23:
24:  VAR
25:    grDriver:integer;
26:    grMode  :integer;
27:    Emcode  :integer;
28:    zangle  :real;
29:    yangle  :real;
30:    angl    :real;
31:    sinz,cosz:real;
32:    x,y,z   :real;
33:    cflg    :integer;
34:    errcode :integer;
35:    x_ax,y_ax,z_ax :array[0..7] of real;
36:    r          :real;
37:    i,st,ed,k,j:integer;
38:    ag         :char;
```

```
  rectang    :array[0..5] of PType;
  weight     :array[0..5] of real;
  index      :array[0..5] of integer;
  header     :string[60];
{*********Draw the polygon in order of looking distance***}
Procedure Draw;
begin

 line(300,20,300,349);
 line(0,160,640,160);
 {****draw the poly in the order that already sorted**}
 for i:=0 to 5 do
   begin
       for j:=0 to 4 do
          begin
          st :=poly[index[i]][j];
          {***add offset and the dimension ratio to the point **}
          rectang[j][0]:=300+trunc(x_ax[st]*82);
          rectang[j][1]:=160-trunc(y_ax[st]*55);
          end;
      setfillpattern(gray1,index[i]);
      fillpoly(5,rectang);
   end;

   {***write the point number on the graphic screen***}
   for i:=0 to 7 do
     begin
       moveto(trunc(300+x_ax[i]*82)+zpoint[i]*10,trunc(160-y_ax[i]*55));
       outtext(chr(i+$30));
     end;
   {***Write the Legend on the graphic screen****}
   moveto(120,10);
   k :=trunc(zangle);
   if zangle<0 then
      begin
        header[17]:='-';
        k:=trunc(-zangle);
      end
   else header[17]:=' ';
   header[18]:=chr((k div 100)+$30); k:=k mod 100;
   header[19]:=chr((k div 10)+$30);  k:=k mod 10;
   header[20]:=chr(k+$30);
   k :=trunc(yangle);
   if yangle<0 then
      begin
        header[37]:='-';
        k:=trunc(-yangle);
      end
   else header[37]:=' ';
   header[38]:=chr((k div 100)+$30); k:=k mod 100;
   header[39]:=chr((k div 10)+$30);  k:=k mod 10;
   header[40]:=chr(k+$30);
   outtext(header);
 end;

 {*****************************************************
  *     Sort the average z distance of the polygon *
  *****************************************************}
 Procedure Sort;
 Var
   i,j,k: integer;
   minindex:integer;
 begin

      {****weight the polygon***}
      for i:=0 to 5 do
        begin
        weight[i] :=0;
        for j:=0 to 3 do
          begin
            st:=poly[i][j];
            weight[i] :=weight[i]+z_ax[st];
```

```
111:            end;
112:          end;
113:
114:        {***sort the weight ****}
115:         for i:=0 to 5 do index[i]:=i;      {initialize pointer}
116:         for i:=0 to 5 do
117:          begin
118:            minindex:=i;
119:            for j:=i+1 to 5 do
120:            begin
121:              if weight[index[minindex]] > weight[index[j]] then
122:                begin
123:                 minindex :=j;
124:                end;
125:            end;
126:          k:=index[i];
127:          index[i]:=index[minindex];
128:          index[minindex]:=k;
129:          end;
130:  end;
131:
132:   {**************************
133:    *   main program        *
134:    **************************}
135:
136:  begin
137:    header :='Z Slant Angle         Y Slant Angle       ';
138:    grDriver:=detect;
139:    initGraph(grdriver,grmode,' ');
140:    errcode:=Graphresult;
141:    if grdriver<3 then
142:      begin
143:        writeln('This program is for graphic adapter that is better than EGA only ');
144:        exit;
145:      end;
146:    setviewport(20,20,600,320,True);
147:    zangle:=0; yangle:=0;
148:    ag:=chr(1);
149:
150:    while ag<>char(27) do       {loop until Esc. is pressed}
151:      begin
152:      {*****calculate the co-ordinate in the looking plane*****}
153:      sinz :=sin(pi*zangle/180);
154:      cosz :=cos(pi*zangle/180);
155:
156:      for i:= 0 to 7 do
157:        begin
158:          x          :=xpoint[i];
159:          y          :=ypoint[i];
160:          z          :=zpoint[i];
161:          r          :=sqrt(x*x+y*y);
162:          if (y = 0) then angl :=90- yangle
163:          else            angl := arctan(x/y)*180/pi-yangle;
164:          x_ax[i]    :=r*sin(pi*angl/180);
165:          y_ax[i]    :=z*sinz +r*cos(pi*angl/180)*cosz;
166:          z_ax[i]    :=z*cosz -r*cos(pi*angl/180)*sinz;
167:       end;
168:
169:    sort;
170:    draw;
171:    { writeln('Zangle ',zangle:5:2,' Y angle ',yangle:5:2);}
172:    while not((ag=chr(0)) or (ag=chr(27))) do ag:=readkey;
173:    ag:=readkey;
174:    if ag=chr(72) then zangle :=zangle-5;    (UP   )
175:    if ag=chr(80) then zangle :=zangle+5;    (DOWN )
176:    if ag=chr(75) then yangle :=yangle-5;    (RIGHT)
177:    if ag=chr(77) then yangle :=yangle+5;    (LEFT )
178:    clearviewport;       {*erase the graph**}
179:    end;
180:    restorecrtmode;
181:  end.
182:
```